A Domain-Specific Language for Rich Motor Skill Architectures

Arne Nordmann and Sebastian Wrede

***Abstract*— Model-driven software development is a promising way to cope with the complexity of system integration in advanced robotics, as it already demonstrated its benefits in domains with comparably challenging system integration requirements. This paper reports on work in progress in this area which aims to improve the research and experimentation process in a collaborative research project developing motor skill architectures for compliant robots. Our goal is to establish a model-driven development process throughout the project around a domain-specific language (DSL) facilitating the compact description of adaptive modular architectures for rich motor skills. Incorporating further languages for other aspects (e.g. mapping to a technical component architecture) the approach allows not only the formal description of motor skill architectures but also automated code-generation for experimentation on technical robot platforms. This paper reports on a first case study exemplifying how the developed AMARSi DSL helps to conceptualize different architectural approaches and to identify their similarities and differences.**

I. INTRODUCTION

The European *AMARSi* Project[1] explores how to extend robot motor skills towards biological richness through the development of modular motor control architectures that thoroughly integrate compliance, learning and adaptive control. A number of inter-disciplinary project partners from Biology, Machine Learning, Systems Engineering and more conduct experimental research to developed the required architectural concepts based on the idea of modeling movement primitives as dynamical systems. Despite this shared approach, a large number of individual architectural designs on how to organize movement primitives existed at the start of the project. Although they all describe systems of the same domain, they are expressed in different notations and blueprints, thus making them hard to compare, see Figure 1.

The discussion about the semantic interpretation of the individual boxologies raised the questions on how to i) enable research on motor control architectures among project partners in a way that shares the same conceptual models and ii) how to link these to reproducable robotics experiments on the compliant robot platforms. Throughout this paper and as the initial use-case for the developed DSL on rich motor skills, we focus on the identification of the domain-specific concepts and its application to improve the interaction among project participants.

Model-driven and domain specific development methods are recognized to cope with the challenges of building complex heterogeneous systems in domains such as aerospace, telecommunication and automotive [1] which face similarly complex integration and modeling challenges as advanced robotics. In the last years, this approach was initially adapted to the robotics domain, for example for task description [2], [3]. These approaches are often based on rather generic domain models or ontologies [4] for semantic modeling of robotics systems and translated into DSLs [5]. Our work in the AMARSi project targets at establishing a model-driven development process focussed on motor control architectures based on movement primitives. To the best of our knowledge no such process or domain-specific language for motor control in advanced robots exists so far. The initial conceptualization achieved in this work mainly considers structural aspects but already helped to understand and discuss the semantics of different architectural elements used by project partners.

The paper is organized as follows: Section II discusses basic requirements for an DSL in the AMARSi motor control domain, based on a domain analysis and a domain-specific meta-model. Based on this analysis, Section III introduces the modularization of DSLs, focusing on the highest level AMARSi DSL, and describing example artifact generation. Section IV shows an use-case of the presented DSL in the AMARSi project before Section V concludes and provides an outlook on challenges and further work.

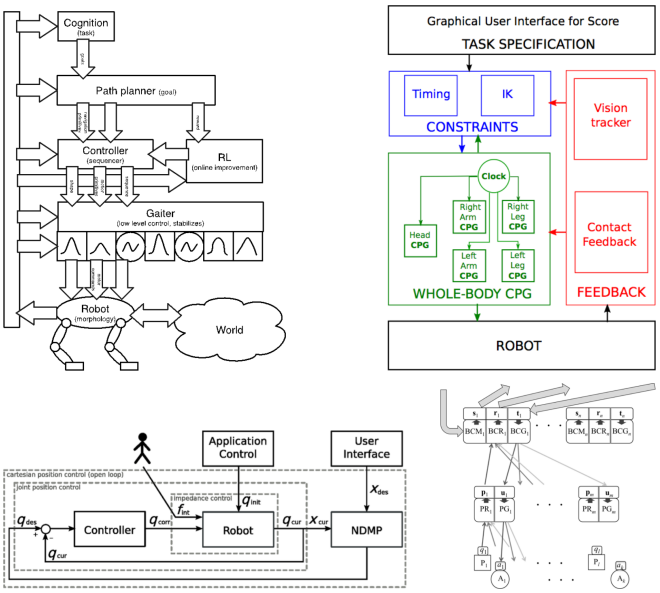

Fig. 1: Different motor control architectures from the AMARSi project expressed in different notations, which makes them hard to compare.

A. Nordmann and S. Wrede are with the Research Institute for Cognition and Robotics, Bielefeld University, P.O. Box 100131, Bielefeld, Germany {anordman,swrede}@cor-lab.uni-bielefeld.de

[1]http://www.amarsi-project.eu

II. APPROACH

Developing domain-specific languages is usually based on a domain analysis to identify concepts and common structures, problems and solutions of an application domain. Large parts of the subsequently presented DSLs are based on an extensive, formal domain analysis we conducted in early phases of the project on the domain of motor skill architectures for compliant robots [6]. The domain we consider here is architectures to generating rich and complex movements through adaptation and learning, both for goal-directed (eg. reaching, foot placement) and periodic movements (eg. walking, locomotion).

As a first step we conducted a formal analysis of the domain of compliant robots control, which led to a detailed view on the control and sensing aspects as well as domain data types of compliant robot control [6]. The domain analysis was based on the formal *Feature-Oriented Domain Analysis* [7] method, resulting in, among other results, detailed feature models for the domain entities.

A second domain analysis was conducted regarding the different architectural approaches within the project (cf. Figure 1). The results of these analysis led to the specification of an architectural meta-model. A meta-model describes the structure and the identified element types of an application domain. An architectural meta-model adds to this perspective important concepts from software architecture which are required to structure, technically orchestrate and deploy complex systems whose overall complexity is well beyond the comprehension of a single developer. The meta-model serves as basis for the domain-specific language development, defining rather generic concepts (e.g. `Components`, `Ports`) and specific ones (e.g. `Adaptive Component`, `Tracking Controller`) [8]. The AMARSi DSL detailed below is based on this meta-model to provide a specific language and syntax dedicated to the AMARSi motor control domain and their solutions, as well as a type-system based on the domain analysis on compliant robot control.

III. AMARSI DOMAIN-SPECIFIC LANGUAGES

Current state of our DSL development divided the meta-model into a set of domain-specific languages, following a *language modularization, extension and composition* approach [9] (LME&C), to separate the DSLs by concerns. The formalization of the meta-model in a domain-specific language will provide an avenue for automation, e.g., for code generation or automatic model verification. The introduced languages cover different – rather technical as well as semantic – aspects of the domain.

The languages were developed together with their projectional editors, using the DSL workbench *Meta-Programming System* [10] (MPS) by *JetBrains*[2]. The projectional editors allow to write systems in concepts, which are *projected* to text for editing purposes [10]. This saves additional parsing steps of the language to extract the concepts from text.

[2]http://www.jetbrains.com/mps/

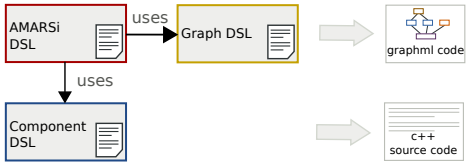

Fig. 2: Language decomposition and artifacts generation from the highest level AMARSi DSL language by using the Component DSL and Graph DSL.

A. Language Modularization

Decomposing the meta-model into a set of domain-specific languages led to the following DSL prototypes:

1) **AMARSi DSL** Expresses motor control systems in the concepts found in the domain analysis, organized around the essential concept of AMARSi `Adaptive Module`. It focuses solely on the domain concepts, hiding the technical aspects. Concepts of this language cover the architectural motor control aspects as well as a type-system and are detailed in the following Section III-B.
2) **Component DSL** Covering the aspect of organizing the system in *components* and their connectivity. Concepts of this language cover component-based software aspects like `Components` with `States`, `Ports`, etc.
3) **Graph DSL** To cover the aspect of *visualization* of complex robotics architectures we follow the already established *GraphML* syntax to express systems in the concepts of `Nodes`, `Edges` and `Subgraphs`, as well as optical aspects such as shapes, colors, etc. The following system visualizations in Figure 4 are rendered based on the generators of this language.

B. AMARSi DSL

The AMARSi DSL is created to allow compact and rich description of motor control architectures in terms of movement primitives using dynamical systems. The AMARSi DSL is created around the essential concepts of the domain, `Adaptive Modules`, `Adaptive Components`, `Spaces` and `Mappings`, with `Adaptive Modules` being the architectural building block to represent a movement primitive. Movement primitives in AMARSi are based on `Dynamical Systems`, potentially adapted and trained with machine learning methods. The essential concepts of the AMARSi DSL are:

- A `Space` is as *"a number of explicit variables that appear to be jointly manipulated or sensed somewhere in our motion control architecture"*, e.g. joint angles of a certain robot limb [8]. The `Types` of `Spaces` are the ones found in the robot control domain analysis introduced before (e.g. `Joint Angles`, `Impedance`).
- An `Adaptive Module` is the main functional building block of the motor control system, representing a movement primitive, containing one or more `Dynamical Systems` and providing an appropriate `Learner` (learning method) for them. It can be

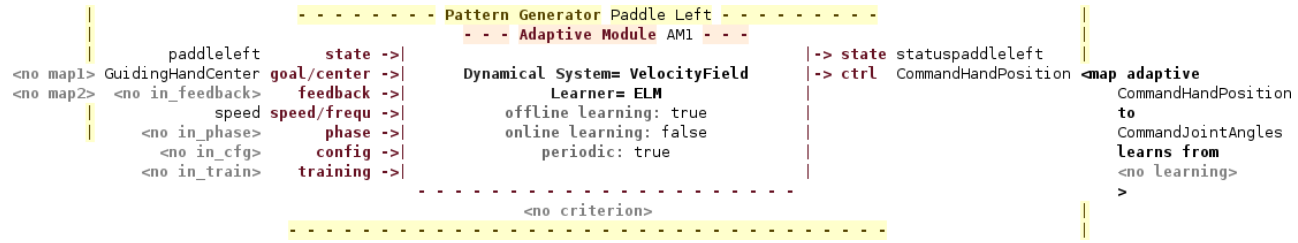

Fig. 3: Example of an Adaptive Component in teh projectional AMARSi DSL editor.

parametrized in shape, speed and goal, its learning methodology (e.g. online, offline) through according input `Spaces`, and can operate closed-loop or open-loop. It defines input `Spaces` for execution and for learning as well as an output space for its control output.

- An `Adaptive Component` is an *"adaptive module together with its input and output Spaces, a basic semantics (control logic) inside the component and timing management"* [8]. It provides the logical structure around an `Adaptive Module`, connecting it with the entire system and managing connections through different states (online learning, offline learning, execution). It can also optionally specify a `Criterion` that indicates, when the movement primitive is finished. The DSL defines several specialized subtypes of `Adaptive Components` found during the domain analysis, like `Tracking Controller`, `Sequencer` or `Pattern Generator`, with specific logical wiring.
- `Mappings` map data between `Spaces` of different `Type` (e.g. `Forward Kinematics`), `Transformations` define the transformation of data between `Spaces` of the same `Type` (e.g. `Coordinate Transformation`).

An example of a movement primitive to perform a paddling movement with the left arm of a humanoid robot is shown in Figure 3. The `Adaptive Module` (red) contains a `Velocity Field` as `Dynamical System`, that is trained with an `Extreme Learning Machine` as `Learner`. The movement can be shaped or adapted for example in its `Speed` or `Goal` as indicated by the input `Spaces` on the left hand side. The `Adaptive Module` is embedded in an `Adaptive Component` (yellow) of the type `Pattern Generator`, which determines the internal wiring of the `Adaptive Module`. It allows to optionally add `Mappings` (in the example at the output) or a stop `Criterion`. The wiring is not visible in the editor, but determines the mapping to the Component DSL and therefore the technical component architecture.

An AMARSi motor control system is modeled by defining `Spaces`, `Mappings` and `Transformations` between them, and connecting `Adaptive Modules` and `Adaptive Components` to the defined `Spaces`. The current state of the projectional editor of the AMARSi DSL allows modeling a system by instantiating and configuring a set of the concepts above.

C. Artifact Generation

Besides modeling systems within the introduced AMARSi DSL, we aim on generating artifacts like visualization and source code from the system model. Figure 2 shows how the introduced DSLs interplay with each other. For system visualization, the AMARSi DSL uses the Graph DSL by providing a generator to map AMARSi DSL concepts to the Graph DSL concepts. The result is a formulation of a system in the concepts of `Nodes` and `Edges`, enriched by optical aspects like coloring. The Graph DSL itself has a generator rendering *graphml* code for visualization purposes.

For generation of executable C++ artifacts, the AMARSi DSL language uses the Component DSL to map the system model to software concepts defined in the Component DSL, like `Components`, `Ports`, etc. to express the component connectivity. Its generator generates C++ code targeting the Compliant Control Architecture (CCA) and Robot Control Interface (RCI) software libraries [11]. It generates executable system files, instantiating and configuring the components and their ports according to the system specified in the AMARSi DSL. Additionally CCA component hulls are generated for implementation, adding typed ports and the component life-cycle including (optionally) online and offline learning hooks.

Rather technical aspects, like the deployment configuration, were explicitly excluded from the AMARSi DSL in the LME&C process, to expose only the motor control aspects to the AMARSi DSL user (the architect). However, MPS allows to not only generate the transitive artifacts (e.g. GraphML code, executable C++ code), but also allows to preserve all intermediate models of the generation process, in our case the Component DSL and Graph DSL models. These intermediate models are generated with either default values or even missing values in the technical aspects, that are under-specified in the AMARSi DSL. Missing and default values can then be added and refined by experts of the particular lower-level DSLs, allowing for example the system administrator to configure the deployment of the motor control system in the Component DSL, or allowing to change visual aspects in the Graph DSL (eg. coloring, highlighting).

IV. APPLICATION

As already stated in the motivation of this work, motor control architectures are expressed at different levels of abstraction, using different notations to express same concepts, and vice versa. This makes comparing different

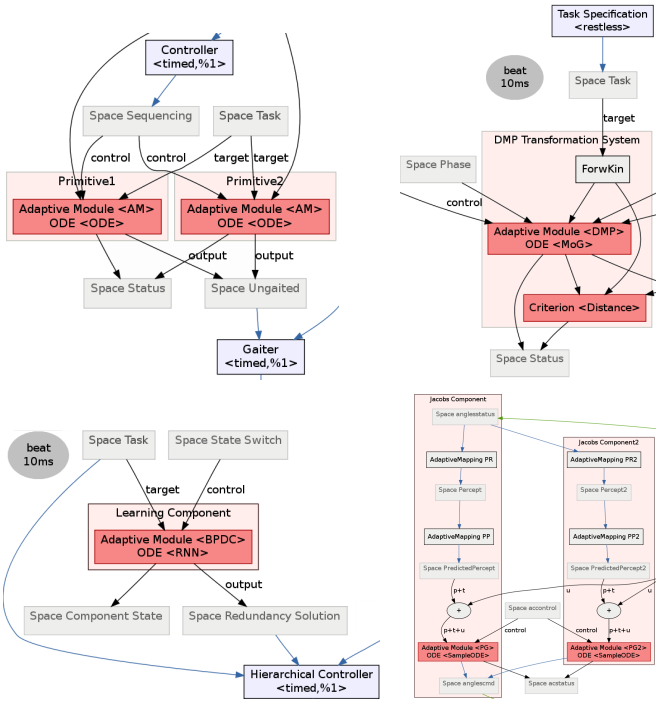

Fig. 4: Excerpt of system visualizations of the four motor control blueprints introduced in Figure 1, expressed in the AMARSi DSL, making different solutions identifiable and comparable.

blueprints especially hard, which is important to enable research on architectural level [8]. Even within a single project we identified at least four different motor control architecture notations, as it was introduced in the beginning and illustrated in Figure 1.

As a first application of the DSL approach and the DSL prototypes introduced before, we modeled the different architecture blueprints of the project in the AMARSi DSL. The four motor control architectures introduced in Figure 1 for example are now comparable and similarities of the approaches, like usage of similar concepts and patterns, and their difference in structure can be identified easier.

Figure 4 shows excerpts of system illustrations of all four approaches, as they were expressed in the AMARSi DSL in terms of `Adaptive Components`, `Adaptive Modules` and `Spaces`. The visualization was created through the Graph DSL as described in Section III-C, generating GraphML code for rendering.

V. CONCLUSION

We presented the work in progress of using domain-specific languages for motor control architectures based on movement primitives in the AMARSi project. We presented, how different notations of architectural blueprints, especially in inter-disciplinary projects, can prevent comparability of different approaches. The AMARSi DSL expresses motor skill architectures based on a domain analysis and the resulting architectural meta-model. As an exemplary use-case we showed how it already helped formulating different blueprints in the same domain terms and notations. Similarities of the different approaches could be extracted easier allowing comparison of the blueprints.

We expect this to be an enabler for architectural research, which was already found plausible in the first applications as presented in Section IV. The presented tool-chain based on a state-of-the-art language workbench will enable us to quickly progress from this starting point towards the core domain aspects of the AMARSi project, namely dynamical systems, learning and adaptive control, eventually facilitating reproducible research on the architectural level.

Further work will continue to establish a model-driven development process in AMARSi around the presented DSLs, extending the C++ code generation to verify our approaches on compliant robot platforms. In terms of further language modularization we will extract the type-system of the AMARSi DSL into a separate *robot data-types* DSL, to also be accessible for applications outside the motor control architecture domain, like the meta-model for data interoperability as presented in [12]. On a conceptual level further development will continue incorporating dynamic aspects of motor skills architectures in the AMARSi DSL.

VI. ACKNOWLEDGEMENT

This work was supported by the European Community's Seventh Framework Programme FP7/2007-2013 Challenge 2 Cognitive Systems, Interaction, Robotics under grant agreement No 248311 - AMARSi.

REFERENCES

[1] A. Van Deursen, P. Klint, and J. Visser. Domain-Specific Languages: An Annotated Bibliography. *ACM Sigplan Notices*, 2000.

[2] I. Pembeci, H. Nilsson, and G. Hager. Functional Reactive Robotics: An Exercise in Principled Integration of Domain-Specific Languages. In *Principles and Practice of Declarative Programming*, pages 168–179. 2002.

[3] R. Simmons and D. Apfelbaum. A Task Description Language for Robot Control. In *International Conference on Intelligent Robots and Systems*, number October, Victoria. B.C., 1998.

[4] G. Lortal, S. Dhouib, and S. Gérard. Integrating Ontological Domain Knowledge into a Robotic DSL. *Lecture Notes in Computer Science*, 6627:401–414, 2011.

[5] L. Zhang, H. Zhang, and Z. Fang. A Domain Specific Architecture Description Language for Autonomous Mobile Robots. In *International Conference on Information and Automation*, number June, pages 283–288, 2012.

[6] A. Nordmann, S. Wrede, N. Tsagarakis, and A. Tuleu. Software Interface for Proprioceptive Sensors and Actuators, 2010. http://www.amarsi-project.eu/system/files/AMARSI-D.7.1.pdf.

[7] S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. FODA: Feature-Oriented Domain Analysis. 1990.

[8] A. Nordmann and S. Wrede. Meta-model and Software Concepts for an Adaptive Component Architecture, 2011. https://www.amarsi-project.eu/system/files/Deliverable-7.3.pdf.

[9] M. Völter. Language and IDE Modularization, Extension and Composition with MPS. In *Generative and Transformational Techniques in Software Engineering*, 2011.

[10] M. Völter. From Programming To Modeling – and back again. *IEEE Software*, 2010.

[11] A. Nordmann, M. Rolf, and S. Wrede. Software Abstractions for Simulation and Control of a Continuum Robot. In *Simulation, Modeling, and Programming for Autonomous Robots*, 2012.

[12] J. Wienke, A. Nordmann, and S. Wrede. A Meta-Model and Toolchain for Improved Interoperability of Robotic Frameworks. In *SIMULATION, MODELING, and PROGRAMMING for AUTONOMOUS ROBOTS*, 2012.